UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/713,633	11/14/2003	Bryan M. Cantrill	03226.342001; SUN040194	7049

32615 7590 01/16/2007
OSHA LIANG L.L.P./SUN
1221 MCKINNEY, SUITE 2800
HOUSTON, TX 77010

EXAMINER	
NGUYEN, PHILLIP H	
ART UNIT	**PAPER NUMBER**
2191	

SHORTENED STATUTORY PERIOD OF RESPONSE	MAIL DATE	DELIVERY MODE
3 MONTHS	01/16/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

If NO period for reply is specified above, the maximum statutory period will apply and will expire 6 MONTHS from the mailing date of this communication.

Office Action Summary	Application No. 10/713,633	Applicant(s) CANTRILL, BRYAN M.	
	Examiner Phillip H. Nguyen	Art Unit 2191	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *14 November 2003*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-36* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-36* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *14 November 2003* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☒ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date *20040205*.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. This action is in response to the original filing of November 14, 2003. Claims 1-36 are pending and have been considered below.

Claim Objections

2. Claims 1 and 19 are objected to because of the following informalities: It recites, "the method for tracing", but there is no tracing performed in the body of the claim. Therefore, "tracing" does not carry patentable weight. Also, the word "for" indicates the intended use and such does not carry patentable weight either. The limitations following the phrase "for" describes only intended use but not necessarily required functionality of the claim and which do not carry patentable weight, which cause the claim to appear as a series of non functional descriptive material/data without any functional relation with each other.

3. Claims 5, 9, 13, 17, 23, 27, 31, and 35 are objected to because of the following informalities: The word "CS" is needed to spell out in the body of the claim.

Appropriate correction is required.

Claim Rejections - 35 USC § 101

4. 35 U.S.C. 101 reads as follows:

> Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent therefor, subject to the conditions and requirements of this title.

5. Claims 1-36 are rejected under 35 U.S.C. 101 because the claimed invention is directed to non-statutory subject matter.

6. Claim 1 is non statutory because the language of the claim raises a question as to whether the outcome of the claim accomplishes a practical application, i.e., produces a useful, concrete, and tangible result. Emulating is normally an abstract idea, but it is acceptable in this case because the phrase "wherein the instruction relates to creating or dismantling a stack frame" is actually performing something here. However, "**creating** or **dismantling** a stack frame" is an abstract idea. Creating or dismantling is just a defining step to define a stack frame, which is not a tangible result because the outcome is not realized as a monitoring or controlling or any other tangible output that would provide a utility. Therefore, the claim is non-statutory. Additional item to consider is the word "for" in the preamble of the claim. It indicates the intended use and such does not carry patentable weight. The limitations following the phrase "for" describes only the intended use but not necessarily required functionality of the claim and which do not carry patentable weight, which cause the claim to appear as a series of non functional descriptive material/data, per se, without any functional relation with each other and therefore, are non-statutory. Claims 2-18 directly or indirectly depend on claim 1 and therefore suffer the same rejection.

Claim 19 is non-statutory because it recites a system but appears reasonable to interpret this system by one of ordinary skill in the art as software, per se. Applicant's specification provides no explicit and deliberate definition of the components ("instrumented program" "a thread", "a trap handler") that make up the system other

than they could be software components, which are directed to functional descriptive material, per se, and are therefore non statutory. Additional item to consider is whether "to emulate the instruction associated with the trap instruction" accomplishes a practical application, i.e., produces a useful, concrete, and tangible result. Emulating process is an abstract idea since it is a pretending process to perform something. The claimed invention fails to define emulating process. Another additional item to consider is the word "for" in the preamble of the claim. It indicates the intended use and such does not carry patentable weight. The limitations following the phrase "for" describes only the intended use but not necessarily required functionality of the claim and which do not carry patentable weight, which cause the claim to appear as a series of non functional descriptive material/data, per se, without any functional relation with each other and therefore, are non-statutory. Claims 20-36 directly or indirectly depend on claim 19 and therefore have been addressed in connection with the rejection set forth in claim 19.

Claim Rejections - 35 USC § 112

7. The following is a quotation of the second paragraph of 35 U.S.C. 112:

> The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

8. Claims 1-36 are rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention.

Claim 1 recites, "the instruction..." is unclear to the examiner as to whether it refers to the trap instruction or instruction. For examining purposes, examiner interprets

it as either trap instruction or instruction. Additional item to consider is whether "the instruction" (**one instruction**) can create or dismantle a stack frame. One of ordinary skill in the art would have known that in order to create or dismantle a stack frame, multiple instructions are needed. Claims 2-18 directly or indirectly depend on claim 1 and therefore have been addressed in connection with the rejection set forth in claim 1.

Claim 19 is a system claim and recites the same limitation as recited in claim 1 and therefore suffers the same reason set forth in claim 1. Claims 20-36 suffer the same rejection as in 19.

Claim Rejections - 35 USC § 102

9. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

10. Claim 1 -36 ~~is~~ are rejected under 35 U.S.C. 102(e) as being anticipated by Yates, Jr. et al. (United States Patent No.: US 7,065,633 B1).

<u>As per claim 1</u>:

Yates discloses a method for tracing an instrumented program, comprising:

- triggering a trap instruction in the instrumented program ("**an exception occurring during execution of a program coded in the CISC instruction set**" Col 23, line 10-11; "**the exception may be a trap requesting a file access service form the CISC operating system on behalf of the program**" Col 23, line 25-27);
- transferring control of the instrumented program to a trap handler associated with the trap instruction ("**the interrupt is delivered to emulator 316, which handles the interrupt**" Col 51, line 39-40); and
- emulating an instruction corresponding to the trap instruction in the trap handler instruction ("**the RISC operating system may include a collection interrupt service routines programmed to emulate instructions in the CISC instruction set**" Col 23, line 13-15; "**when a high-priority X86 interrupt interrupts X86 emulator 316 while emulating a complex instruction…the interrupt is delivered to emulator 316, which handles the interrupt**" Col 51, line 34-42), wherein the instruction relates to creating or dismantling a stack frame instruction ("**if the program creates code in writable storage (stack or heap) on the fly**" Col 40, line 16-17).

As per claim 2:

Yates discloses the method as in claim 1 above; and further discloses:

- replacing the instruction with the trap instruction in the instrumented program ("**Probing**" Col 30, line 24, **probing is a process of replacing instruction with probe instruction for monitoring the program flow**).

As per claim 3:

Yates discloses the method as in claim 1 above; and further discloses:

- calling into a tracing framework to execute an action associated with the trap instruction ("**during emulation of the X86 program, prober 600 monitors the program flow for execution of X86 instructions that have been translated into native code**" Col 32, line 7-9).

As per claim 4:

Yates discloses the method as in claim 1 above; and further discloses:

- wherein emulating the instruction comprises emulating a push1 instruction ("**any asynchronous interrupts that occur during the PUSHA converter recipe are handled in tapestry operating system 312 or in emulator 316**" Col 132, line 23-25).

As per claim 5:

Yates discloses the method as in claim 4 above; and further discloses wherein emulating a push1 instruction comprises:

- obtaining a stack pointer location, wherein the stack pointer location corresponds to a location in the stack frame ("**the eight STOREDEC instructions 951 push the eight general purpose registers EAX, ECX, EDX, EBX, ESP, EBP, ESI and ECI onto the stack**" Col 132, line 7-9);
- incrementing an instruction pointer to obtain an incremented instruction pointer ("**the instruction pointer (IP) value is incremented by the length count after each instruction that is marked with an end-of-recipe marker**" Col 122, line 38-40);
- loading the incremented instruction pointer in the stack frame at one location before the stack pointer location ("**STOREDEC, is a pre-decrementing store that pushes the IP value onto the stack segment (SS) at the offset specified by the stack pointer ESP**" Col 136, line 7-10);
- loading a code segment (CS) value stored one location after the stack pointer location into the stack pointer location ("**the current offset into the code segment (EIP) is pushed onto the stack**" Col 135, line 48-49);
- loading an EFLAGS values stored two locations after the stack pointer location into one location after the stack pointer ("**whether addressing is physical or virtual, the floating point stack pointer, and the full/empty state of floating-point register, and operand sizes are encoded in segment descriptors, the EFLAGS register**" Col 91, line 13-16); and
- loading a base pointer into two location after the stack pointer location ("**the eight STOREDEC instructions 951 push the eight general purpose**

registers EAX, ECX, EDX, EBX, ESP, EBP, ESI and ECI onto the stack" Col 132, line 7-9).

As per claim 6:

Yates discloses the method as in claim 5 above; and further discloses:

- decrementing the stack pointer location by one location ("**the register carries the intermediate decrementing of the stack pointer**" Col 132, line 43-44); and
- issuing a return from interrupt instruction ("**a trap instruction transfers control to an exception handler**" Col 94, line 15-16).

As per claim 7:

Yates discloses the method as in claim 5 above; and further discloses:

- wherein the instruction pointer is loaded at the stack pointer location **STOREDEC, is a pre-decrementing store that pushes the IP value onto the stack segment (SS) at the offset specified by the stack pointer ESP**" Col 136, line 7-10).

As per claim 8:

Yates discloses the method as in claim 1 above; and further discloses:

- emulating the instruction comprises emulating a enter instruction ("**PUSHA**" Col 132, line 36; **MOV instruction**" Col 132, line 41).

As per claim 9:

Yates discloses the method as in claim 8 above; and further discloses:

- obtaining a stack pointer location, wherein the stack pointer location is corresponding to a location in the stack frame ("**the eight STOREDEC instructions 951 push the eight general purpose registers EAX, ECX, EDX, EBX, ESP, EBP, ESI and ECI onto the stack**" Col 132, line 7-9);
- incrementing an instruction pointer to obtain an incremented instruction pointer ("**the instruction pointer (IP) value is incremented by the length count after each instruction that is marked with an end-of-recipe marker**" Col 122, line 38-40);
- loading the incremented instruction pointer in the stack frame at one location before the stack pointer location ("**STOREDEC, is a pre-decrementing store that pushes the IP value onto the stack segment (SS) at the offset specified by the stack pointer ESP**" Col 136, line 7-10);
- loading a code segment (CS) value stored one location after the stack pointer location into the stack pointer location ("**the current offset into the code segment (EIP) is pushed onto the stack**" Col 135, line 48-49);
- loading an EFLAGS values stored two locations after the stack pointer location into one location after the stack pointer ("**whether addressing is physical or virtual, the floating point stack pointer, and the full/empty state of floating-point register, and operand sizes are encoded in segment descriptors, the EFLAGS register**" Col 91, line 13-16);

- loading a base pointer into two location after the stack pointer location ("**the eight STOREDEC instructions 951 push the eight general purpose registers EAX, ECX, EDX, EBX, ESP, EBP, ESI and ECI onto the stack**" Col 132, line 7-9); and
- loading the base pointer into a base pointer register ("**EBP register of the X86 architecture are mapped by converter hardware 136 to register R53**" Col 34, line 9-10).

As per claim 10:

Yates discloses the method as in claim 9 above; and further discloses:

- decrementing the stack pointer location by one location ("**the register carries the intermediate decrementing of the stack pointer**" Col 132, line 43-44); and
- issuing a return from interrupt instruction ("**a trap instruction transfers control to an exception handler**" Col 94, line 15-16).

As per claim 11:

Yates discloses the method as in claim 9 above; and further discloses:

- wherein the instruction pointer is loaded at the stack pointer location (**STOREDEC, is a pre-decrementing store that pushes the IP value onto the stack segment (SS) at the offset specified by the stack pointer ESP**" Col 136, line 7-10).

<u>As per claim 12</u>:

Yates discloses the method as in claim 1 above; and further discloses:

- wherein emulating the instruction comprises emulating a pop1 instruction ("**when an X86 POPF instruction is emulated**" Col 58, line 1).

<u>As per claim 13</u>:

Yates discloses the method as in claim 12 above; and further discloses:

- obtaining a stack pointer location, wherein the stack pointer location corresponds to a location in the stack frame ("**the eight STOREDEC instructions 951 push the eight general purpose registers EAX, ECX, EDX, EBX, ESP, EBP, ESI and ECI onto the stack**" Col 132, line 7-9);
- loading the base pointer obtained from three location after the stack pointer location into a base pointer register ("**EBP register of the X86 architecture are mapped by converter hardware 136 to register R53**" Col 34, line 9-10);
- loading an EFLAGS values stored two locations after the stack pointer location into one location after the stack pointer ("**whether addressing is physical or virtual, the floating point stack pointer, and the full/empty state of floating-point register, and operand sizes are encoded in segment descriptors, the EFLAGS register**" Col 91, line 13-16);
- loading a code segment (CS) value stored one location after the stack pointer location into the stack pointer location ("**the current offset into the code segment (EIP) is pushed onto the stack**" Col 135, line 48-49);

- incrementing an instruction pointer to obtain an incremented instruction pointer ("**the instruction pointer (IP) value is incremented by the length count after each instruction that is marked with an end-of-recipe marker**" Col 122, line 38-40); and
- loading the incremented instruction pointer in the stack frame at one location before the stack pointer location ("**STOREDEC, is a pre-decrementing store that pushes the IP value onto the stack segment (SS) at the offset specified by the stack pointer ESP**" Col 136, line 7-10).

As per claim 14:

Yates discloses the method as in claim 13 above; and further discloses:

- incrementing the stack pointer location by one location ("**when an X86 program changes its stack, both the stack segment descriptor (SS) and the offset may need to be changed**" Col 144, line 65-67; "…**the instruction that loads the stack offset into the stack pointer**" Col 145, line 2); and
- issuing a return from interrupt instruction ("**a trap instruction transfers control to an exception handler**" Col 94, line 15-16).

As per claim 15:

Yates discloses the method as in claim 13 above; and further discloses:

- wherein the instruction pointer is loaded at the stack pointer location **STOREDEC, is a pre-decrementing store that pushes the IP value onto**

the stack segment (SS) at the offset specified by the stack pointer ESP" Col 136, line 7-10).

As per claim 16:

Yates discloses the method as in claim 1 above; and further discloses:

- wherein emulating the instruction comprises emulating a leave instruction ("**at the end of emulating the move or pop instruction**" Col 145, line 18).

As per claim 17:

Yates discloses the method as in claim 16 above; and further discloses:

- obtaining a stack pointer location, wherein the stack pointer location corresponds to a location in the stack frame ("**the eight STOREDEC instructions 951 push the eight general purpose registers EAX, ECX, EDX, EBX, ESP, EBP, ESI and ECI onto the stack**" Col 132, line 7-9);
- loading the base pointer obtained at the base pointer location into base pointer register ("**EBP register of the X86 architecture are mapped by converter hardware 136 to register R53**" Col 34, line 9-10);
- loading an EFLAGS values stored two locations after the stack pointer location into one location after the stack pointer ("**whether addressing is physical or virtual, the floating point stack pointer, and the full/empty state of floating-point register, and operand sizes are encoded in segment descriptors, the EFLAGS register**" Col 91, line 13-16);

- loading a code segment (CS) value stored one location after the stack pointer location into the stack pointer location ("**the current offset into the code segment (EIP) is pushed onto the stack**" Col 135, line 48-49);
- incrementing an instruction pointer to obtain an incremented instruction pointer ("**the instruction pointer (IP) value is incremented by the length count after each instruction that is marked with an end-of-recipe marker**" Col 122, line 38-40);
- loading the incremented instruction pointer in the stack frame at one location before the stack pointer location ("**STOREDEC, is a pre-decrementing store that pushes the IP value onto the stack segment (SS) at the offset specified by the stack pointer ESP**" Col 136, line 7-10); and
- loading the instruction pointer at three location before the base pointer ("**EIP is pushed onto the stack**" Col 135, line 49).

<u>As per claim 18</u>:

Yates discloses the method as in clam 17 above; and further discloses:

- setting the stack pointer location to three locations before the base pointer location ("**the eight STOREDEC instructions 951 push the eight general purpose registers EAX, ECX, EDX, EBX, ESP, EBP, ESI and ECI onto the stack**" Col 132, line 7-9, **the idea is to obtain the stack pointer**);
- incrementing the stack pointer location by one location ("**when an X86 program changes its stack, both the stack segment descriptor (SS) and**

the offset may need to be changed" Col 144, line 65-67; "...**the instruction that loads the stack offset into the stack pointer**" Col 145, line 2); and

- issuing a return from interrupt instruction ("**a trap instruction transfers control to an exception handler**" Col 94, line 15-16).

<u>As per claim 19</u>:

Yates discloses a system for tracing an instrumented program, comprising:

- the instrumented program comprising at least one trap instruction associated with an instruction ("**a trap instruction**" Col 94, line 15), wherein the instruction relates to creating or dismantling a stack frame ("**if the program creates code in writable storage (stack or heap) on the fly**" Col 40, line 16-17).
- a thread configured to execute the instrumented program ("**the operating system and the interrupted thread may execute in different instruction set architecture of the computer**" Col 6, line 19-20); and
- a trap handler ("**trap handler**" Col 125, line 52) configured to halt execution of the thread when the trap instruction is encountered and to emulate the instruction associated with the trap instruction (**the RISC operating system may include a collection interrupt service routines programmed to emulate instructions in the CISC instruction set**" Col 23, line 13-15; "**when a high-priority X86 interrupt interrupts X86 emulator 316 while emulating**

a complex instruction...the interrupt is delivered to emulator 316, which handles the interrupt" Col 51, line 34-42).

<u>As per claim 20</u>:

Yates discloses the system as in claim 19 above; and further discloses:

- a tracing framework operatively connected to the trap handler configured to perform an action associated with the trap instruction ("**during emulation of the X86 program, prober 600 monitors the program flow for execution of X86 instructions that have been translated into native code**" Col 32, line 7-9).

<u>As per claim 21</u>:

Yates discloses the system as in claim 19 above; and further discloses:

- wherein the instruction is replaced with a trap instruction in the instrumented program ("**Probing**" Col 30, line 24, **probing is a process of replacing instruction with probe/trap instruction for monitoring the program flow**).

<u>As per claim 22</u>:

Yates discloses the system as in claim 19 above; and further discloses:

- wherein emulating the instruction comprises emulating a push1 instruction ("**any asynchronous interrupts that occur during the PUSHA converter**

recipe are handled in tapestry operating system 312 or in emulator 316" Col 132, line 23-25).

As per claim 23:

Yates discloses the system as in claim 22 above; and further discloses:

- obtaining a stack pointer location, wherein the stack pointer location corresponds to a location in the stack frame ("**the eight STOREDEC instructions 951 push the eight general purpose registers EAX, ECX, EDX, EBX, ESP, EBP, ESI and ECI onto the stack**" Col 132, line 7-9);
- incrementing an instruction pointer to obtain an incremented instruction pointer ("**the instruction pointer (IP) value is incremented by the length count after each instruction that is marked with an end-of-recipe marker**" Col 122, line 38-40);
- loading the incremented instruction pointer in the stack frame at one location before the stack pointer location ("**STOREDEC, is a pre-decrementing store that pushes the IP value onto the stack segment (SS) at the offset specified by the stack pointer ESP**" Col 136, line 7-10);
- loading a code segment (CS) value stored one location after the stack pointer location into the stack pointer location ("**the current offset into the code segment (EIP) is pushed onto the stack**" Col 135, line 48-49);
- loading an EFLAGS values stored two locations after the stack pointer location into one location after the stack pointer ("**whether addressing is**

physical or virtual, the floating point stack pointer, and the full/empty state of floating-point register, and operand sizes are encoded in segment descriptors, the EFLAGS register" Col 91, line 13-16); and

- loading a base pointer into two location after the stack pointer location ("**the eight STOREDEC instructions 951 push the eight general purpose registers EAX, ECX, EDX, EBX, ESP, EBP, ESI and ECI onto the stack**" Col 132, line 7-9).

As per claim 24:

Yates discloses the system as in claim 23 above; and further discloses:

- decrementing the stack pointer location by one location ("**the register carries the intermediate decrementing of the stack pointer**" Col 132, line 43-44); and
- issuing a return from interrupt instruction ("**a trap instruction transfers control to an exception handler**" Col 94, line 15-16).

As per claim 25:

Yates discloses the system as in claim 23 above; and further discloses:

- wherein the instruction pointer is loaded at the stack pointer location **STOREDEC, is a pre-decrementing store that pushes the IP value onto the stack segment (SS) at the offset specified by the stack pointer ESP**" Col 136, line 7-10).

<u>As per claim 26</u>:

Yates discloses the system as in claim 19 above; and further discloses:

- emulating the instruction comprises emulating a enter instruction ("**PUSHA**" Col 132, line 36; **MOV instruction**" Col 132, line 41).

<u>As per claim 27</u>:

Yates discloses the system as in claim 26 above; and further discloses:

- obtaining a stack pointer location, wherein the stack pointer location is corresponding to a location in the stack frame ("**the eight STOREDEC instructions 951 push the eight general purpose registers EAX, ECX, EDX, EBX, ESP, EBP, ESI and ECI onto the stack**" Col 132, line 7-9);
- incrementing an instruction pointer to obtain an incremented instruction pointer ("**the instruction pointer (IP) value is incremented by the length count after each instruction that is marked with an end-of-recipe marker**" Col 122, line 38-40);
- loading the incremented instruction pointer in the stack frame at one location before the stack pointer location ("**STOREDEC, is a pre-decrementing store that pushes the IP value onto the stack segment (SS) at the offset specified by the stack pointer ESP**" Col 136, line 7-10);
- loading a code segment (CS) value stored one location after the stack pointer location into the stack pointer location ("**the current offset into the code segment (EIP) is pushed onto the stack**" Col 135, line 48-49);

- loading an EFLAGS values stored two locations after the stack pointer location into one location after the stack pointer ("**whether addressing is physical or virtual, the floating point stack pointer, and the full/empty state of floating-point register, and operand sizes are encoded in segment descriptors, the EFLAGS register**" Col 91, line 13-16);
- loading a base pointer into two location after the stack pointer location ("**the eight STOREDEC instructions 951 push the eight general purpose registers EAX, ECX, EDX, EBX, ESP, EBP, ESI and ECI onto the stack**" Col 132, line 7-9); and
- loading the base pointer into a base pointer register ("**EBP register of the X86 architecture are mapped by converter hardware 136 to register R53**" Col 34, line 9-10).

<u>As per claim 28</u>:

Yates discloses the system as in claim 27 above; and further discloses:

- decrementing the stack pointer location by one location ("**the register carries the intermediate decrementing of the stack pointer**" Col 132, line 43-44); and
- issuing a return from interrupt instruction ("**a trap instruction transfers control to an exception handler**" Col 94, line 15-16).

As per claim 29:

Yates discloses the system as in claim 27 above; and further discloses:

- wherein the instruction pointer is loaded at the stack pointer location (**STOREDEC, is a pre-decrementing store that pushes the IP value onto the stack segment (SS) at the offset specified by the stack pointer ESP**" Col 136, line 7-10).

As per claim 30:

Yates discloses the system as in claim 19 above; and further discloses:

- wherein emulating the instruction comprises emulating a pop1 instruction ("**when an X86 POPF instruction is emulated**" Col 58, line 1).

As per claim 31:

Yates discloses the system as in claim 30 above; and further discloses:

- obtaining a stack pointer location, wherein the stack pointer location corresponds to a location in the stack frame ("**the eight STOREDEC instructions 951 push the eight general purpose registers EAX, ECX, EDX, EBX, ESP, EBP, ESI and ECI onto the stack**" Col 132, line 7-9);
- loading the base pointer obtained from three location after the stack pointer location into a base pointer register ("**EBP register of the X86 architecture are mapped by converter hardware 136 to register R53**" Col 34, line 9-10);

- loading an EFLAGS values stored two locations after the stack pointer location into one location after the stack pointer ("**whether addressing is physical or virtual, the floating point stack pointer, and the full/empty state of floating-point register, and operand sizes are encoded in segment descriptors, the EFLAGS register**" Col 91, line 13-16);
- loading a code segment (CS) value stored one location after the stack pointer location into the stack pointer location ("**the current offset into the code segment (EIP) is pushed onto the stack**" Col 135, line 48-49);
- incrementing an instruction pointer to obtain an incremented instruction pointer ("**the instruction pointer (IP) value is incremented by the length count after each instruction that is marked with an end-of-recipe marker**" Col 122, line 38-40); and
- loading the incremented instruction pointer in the stack frame at one location before the stack pointer location ("**STOREDEC, is a pre-decrementing store that pushes the IP value onto the stack segment (SS) at the offset specified by the stack pointer ESP**" Col 136, line 7-10).

<u>As per claim 32</u>:

Yates discloses the system as in claim 31 above; and further discloses:

- incrementing the stack pointer location by one location ("**when an X86 program changes its stack, both the stack segment descriptor (SS) and**

the offset may need to be changed" Col 144, line 65-67; "...**the instruction that loads the stack offset into the stack pointer**" Col 145, line 2); and

- issuing a return from interrupt instruction ("**a trap instruction transfers control to an exception handler**" Col 94, line 15-16).

<u>As per claim 33</u>:

Yates discloses the system as in claim 31 above; and further discloses:

- wherein the instruction pointer is loaded at the stack pointer location **STOREDEC, is a pre-decrementing store that pushes the IP value onto the stack segment (SS) at the offset specified by the stack pointer ESP**" Col 136, line 7-10).

<u>As per claim 34</u>:

Yates discloses the system as in claim 19 above; and further discloses:

- wherein emulating the instruction comprises emulating a leave instruction ("**at the end of emulating the move or pop instruction**" Col 145, line 18).

<u>As per claim 35</u>:

Yates discloses the system as in claim 34 above; and further discloses:

- obtaining a stack pointer location, wherein the stack pointer location corresponds to a location in the stack frame ("**the eight STOREDEC**

instructions 951 push the eight general purpose registers EAX, ECX, EDX, EBX, ESP, EBP, ESI and ECI onto the stack" Col 132, line 7-9);

- loading the base pointer obtained at the base pointer location into base pointer register ("**EBP register of the X86 architecture are mapped by converter hardware 136 to register R53**" Col 34, line 9-10);
- loading an EFLAGS values stored two locations after the stack pointer location into one location after the stack pointer ("**whether addressing is physical or virtual, the floating point stack pointer, and the full/empty state of floating-point register, and operand sizes are encoded in segment descriptors, the EFLAGS register**" Col 91, line 13-16);
- loading a code segment (CS) value stored one location after the stack pointer location into the stack pointer location ("**the current offset into the code segment (EIP) is pushed onto the stack**" Col 135, line 48-49);
- incrementing an instruction pointer to obtain an incremented instruction pointer ("**the instruction pointer (IP) value is incremented by the length count after each instruction that is marked with an end-of-recipe marker**" Col 122, line 38-40);
- loading the incremented instruction pointer in the stack frame at one location before the stack pointer location ("**STOREDEC, is a pre-decrementing store that pushes the IP value onto the stack segment (SS) at the offset specified by the stack pointer ESP**" Col 136, line 7-10); and

- loading the instruction pointer at three location before the base pointer ("**EIP is pushed onto the stack**" Col 135, line 49).

As per claim 36:

Yates discloses the system as in claim 35 above; and further discloses:

- setting the stack pointer location to three locations before the base pointer location ("**the eight STOREDEC instructions 951 push the eight general purpose registers EAX, ECX, EDX, EBX, ESP, EBP, ESI and ECI onto the stack**" Col 132, line 7-9, **the idea is to obtain the stack pointer**);
- incrementing the stack pointer location by one location ("**when an X86 program changes its stack, both the stack segment descriptor (SS) and the offset may need to be changed**" Col 144, line 65-67; "...**the instruction that loads the stack offset into the stack pointer**" Col 145, line 2); and
- issuing a return from interrupt instruction ("**a trap instruction transfers control to an exception handler**" Col 94, line 15-16).

Conclusion

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Phillip H. Nguyen whose telephone number is (571) 270-1070. The examiner can normally be reached on Monday - Thursday 10:00 AM - 3:00 PM EST.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Wei Y. Zhen can be reached on (571) 272-3708. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

PN
01/03/2007

WEI ZHEN
SUPERVISORY PATENT EXAMINER